„It's all greek to me!“
Verschlüsselung mit dem RSA-Verfahren

Markus Paul

Es war eine kleine Sensation, als am 08. Nov. 2005 einem Team von deutschen Informatikern nach dreimonatiger Arbeit mit 80 Computern mit 2,2-GHz-CPU gelang, die Zahl RSA-640 mit 193 Dezimalstellen zu faktorisieren:

3107418240490043721350750035888567930037346022842
7275457201619488232064405180815045563468296717232
8678243791627283803341547107310850191954852900733
7724822783525742386454014691736602477652346609

Die beiden Faktoren sind:
1634733'6458092538'4844313388'3865090859'8417836700'
3309231218'1110852389'3331001045'0815121211'8167511
579 und
1900871'2816648221'1312685157'3935413975'4718967899'
6851549366'6638539088'0271038021'0449895719'1261465
571.

Für diese Faktorisierung hatte die Firma RSA Security (www.rsasecurity.com) ein Preisgeld von $ 20 000 ausgesetzt, denn auf der Schwierigkeit, große Zahlen in ihre Primfaktoren zu zerlegen, beruht die Sicherheit des RSA-Verfahrens. Ob Bankautomat, Handy, Kreditkartenüberweisung im Internet, Verschlüsselung von E-Mails, überall wird mit RSA verschlüsselt.

Wie funktioniert diese Schlüsseltechnologie? Kann RSA mit technologischer Unterstützung im Unterricht behandelt werden?

RSA ist ein asymmetrisches Kryptosystem, mit dem Nachrichten mit einem öffentlichen Schlüssel (public key) verschlüsselt und mit einem privaten Schlüssel (private key) entschlüsselt werden. Man kann die asymmetrische Verschlüsselung mit einem Briefkasten vergleichen: Jeder kann einen Brief einwerfen, aber nur der Eigentümer des Briefkastens kann den Brief lesen. Das Verfahren beruht auf zahlentheoretischen Sätzen und funktioniert folgendermaßen:

Schlüsselerzeugung:

1. Wähle zufällig zwei große Primzahlen p und q und berechne den Modul n = p·q und phi = (p-1)·(q-1).
2. Wähle eine Verschlüsselungszahl e (encryption) mit 1 < e < n und ggT(e, phi) = 1 (e teilerfremd zu phi)
3. Berechne eine Entschlüsselungszahl d (decryption) so, dass $e \cdot d \equiv 1 \bmod phi$, d.h. e·d-1 teilt phi. (Probieren oder euklidischer Algorithmus)
4. Gib das Paar (n,e) als öffentlichen Schlüssel (public key) bekannt, halte das Paar (n,d) als privaten Schlüssel (private key) geheim.

Verschlüsseln:
Chiffriere einen Klartext m (message) zum Geheimtext c (cipher text) durch $c = m^e \bmod n$.

Entschlüsseln:
Dechiffriere den Geheimtext zum Klartext durch $m = c^d \bmod n$.

Wir demonstrieren dieses Verfahren an einem Beispiel mit Primzahlen, die etwas kleiner sind als in der Praxis:

Frau G.Heim möchte ein virtuelles Postfach einrichten, auf das nur sie Zugriff hat. Sie erzeugt einen öffentlichen und einen privaten Schlüssel in vier Schritten:

1. Schritt: Sie wählt zufällig zwei Primzahlen, p = 3 und q = 5 (in der Praxis haben diese Primzahlen jeweils mindestens 200 Stellen). Das Produkt ergibt den Modul n = p·q = 15. Weiters berechnet sie die Zahl phi = (p-1)·(q-1) = 8

2. Schritt: Frau G.Heim sucht eine weitere Zahl e (encryption, Verschlüsselung) mit 1 < e < n und ggT(e,phi) = 1, d.h. e und phi müssen relativ prim sein. Die kleinste Zahl, die diese Bedingung erfüllt, ist e = 3

3. Schritt: Frau G.Heim berechnet eine Zahl d (decryption, Entschlüsselung) mit der Eigenschaft:
$e \cdot d \equiv 1 \bmod phi$, also $3 \cdot d \equiv 1 \bmod 8$; d = 11 erfüllt diese Eigenschaft. d kann mit dem euklidischen Algorithmus berechnet oder durch Probieren ermittelt werden:

$3 \cdot d - 1 = s \cdot 8$ bzw $(3 \cdot d - 1)/8$ ist ganzzahlig

Die kleinsten Zahlen, die ein ganzzahliges Ergebnis liefern, sind 3 und 11. Da d nicht gleich e sein sollte, wählt Frau G.Heim d = 11. Beachten Sie: d kann nur dann ermittelt werden, wenn man die Faktoren p und q von n kennt.

4. Schritt: Frau G.Heim gibt den öffentlichen Schlüssel (public key) (n, e) = (15, 3) bekannt; den privaten Schlüssel (private key) (n, d) = (15, 11) hält sie geheim.

Verschlüsselung:
Herr J.Dermann will nun eine Nachricht m (message) verschlüsselt an Frau G.Heim schicken.

Die Nachricht „`hilfe`“ verschlüsselt er in Zahlenblöcken der Länge 2, die Zahlen geben die Stellung des entsprechenden Buchstabens im Alphabet an. Die Zahlenblöcke müssen kleiner als n sein:

08 09 12 06 05.

Er besorgt sich den öffentlichen Schlüssel und ermittelt den Geheimtext c (cipher text) mit $c \equiv m^e \bmod n$

$8^3 = 512 \equiv 2 \bmod 15$
$9^3 = 729 \equiv 9 \bmod 15$
$12^3 = 1728 \equiv 3 \bmod 15$
$6^3 = 216 \equiv 6 \bmod 15$
$5^3 = 125 \equiv 5 \bmod 15$

Der Geheimtext c (cipher text) lautet: 02 09 03 06 05.

Entschlüsselung:
Nur Frau G.Heim kann diesen Cipher-Text mit dem privaten Schlüssel d = 11 entschlüsseln mit der diskreten Exponentialfunktion: $m \equiv c^d \bmod n$

$2^{11} = 2048 \equiv 8 \bmod 15$
$9^{11} = 31381059609 \equiv 9 \bmod 15$
$3^{11} = 177147 \equiv 12 \bmod 15$

$6^{11} =$ 362797056 ≡ 6 mod 15
$5^{11} =$ 48828125 ≡ 5 mod 15

Der Klartext lautet: 08 09 12 06 05

Bei diesen Berechnungen treten zwei numerische Probleme auf:

- Selbst bei kleinen Modul-Zahlen erhalten wir bei der diskreten Exponentialfunktion sehr große Zahlen, die sich nur durch ein CAS oder durch rekursive Restbildung bewältigen lassen.
- Zur Verschlüsselungszahl e muss mit dem erweiterten euklidischen Algorithmus das modulare Inverse, die Entschlüsselungszahl d berechnet werden, sodass gilt:
  e·d ≡ 1 mod phi,

RSA mit Voyage 200

Für die Berechnung der Reste steht die Funktion `mod(`*Zahl*`,`*Modul*`)` zur Verfügung, die sehr leistungsfähig ist.

Die Funktion `privkey(phi,e)` berechnet aus phi und dem öffentlichen Schlüssel e mit dem erweiterten euklidischen Algorithmus den privaten Schlüssel d: (Eingabe über APPS 7 (Programm Editor) 3 (New...); Type: 2:Function; Variable: privkey)

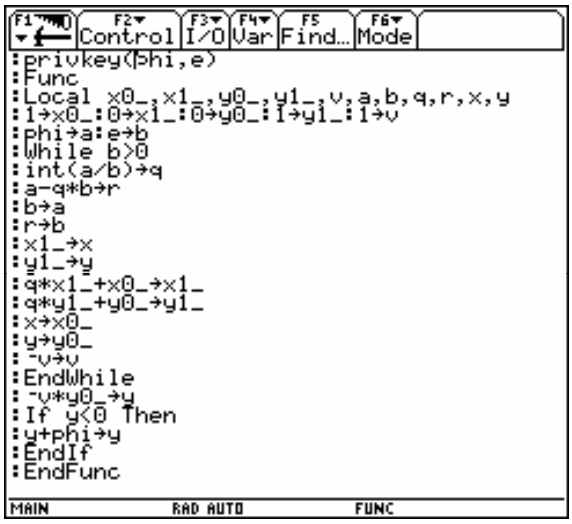

Abb. 1

Mit diesen beiden Funktionen kann verschlüsselt und entschlüsselt werden:

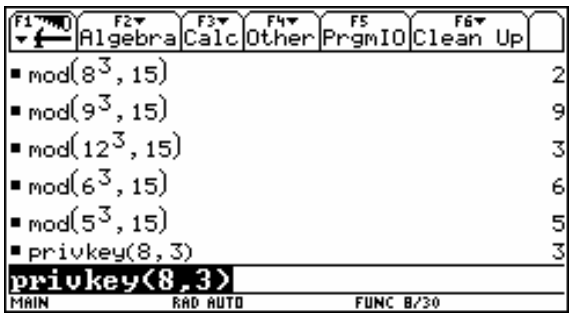

Abb.2

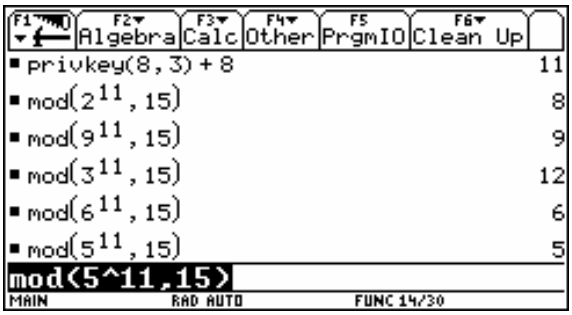

Abb. 3

RSA mit TI-83/84 Plus

Am TI-83/84 Plus steht die Funktion `mod()` nicht zur Verfügung. Man muss sich mit einem kleinen Programm `modexp.83p` behelfen, das b^e mod n rekursiv berechnet:

```
Input "BASIS B:",B
Input "EXPONENT E:",E
Input "MODUL N:",N
B→Z
For(I,2,E)
BZ-int(BZ/N)*N→Z
End
Disp "B^E MOD N=",Z
```

Abb. 4

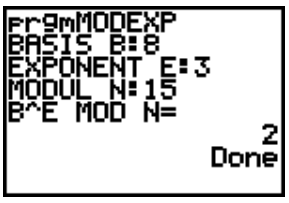

Abb. 5

Das Programm `privkey.83p` berechnet mit dem erweiterten euklidischen Algorithmus aus phi und e den privaten Schlüssel d: (Eingabe über (NEW) , Name = PRIVKEY)

```
Input "PHI:",A
Input "E:",B
{1,0}→ʟX
{0,1}→ʟY
1→V
While B≠0
iPart(A/B)→Q
A-QB→R
B→A
R→B
ʟX(2)→X
ʟY(2)→Y
QʟX(2)+ʟX(1)→ʟX(2)
QʟY(2)+ʟY(1)→ʟY(2)
X→ʟX(1)
Y→ʟY(1)
-V→V
End
VʟX(1)→X
-VʟY(1)→Y
Disp "GGT:",A
Disp "D:",Y
```

Abb.6

Abb. 7

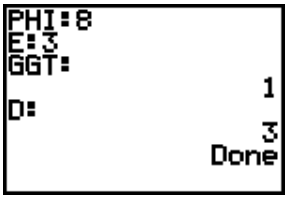

Abb. 8

RSA mit TI InterActive!™

Mit TI InterActive!™ lässt sich das RSA-Verfahren sehr elegant und übersichtlich mit Listen realisieren:

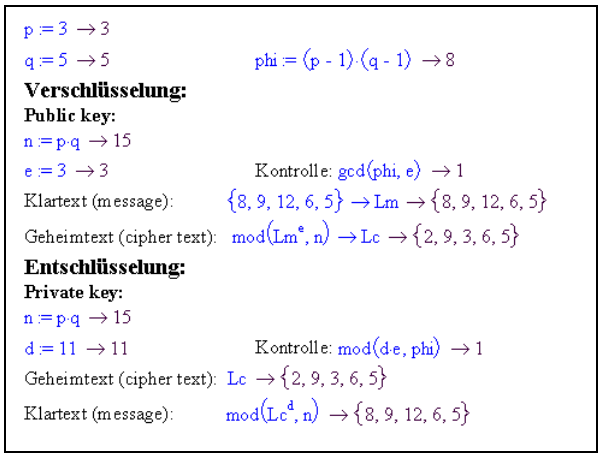

Abb. 9

Auch mit TI InterActive!™ lässt sich eine Funktion **privkey** definieren:

```
Define privkey(phi,e) = Func
 :: Local a,b,x,x1,x2,y,y1,y2,v
 :: a := phi :: b := e :: x1:=1 :: x2:=0 :: y1:=0 :: y2:=1 :: v:=1
 :: While b>0
 :: q := int(a/b) :: r:=a-q*b :: a := b :: b:=r
 :: x:=x2 :: y:=y2 :: x2:=q*x2+x1 :: y2:=q*y2+y1
 :: x1:=x :: y1:=y :: v:=-v
 :: EndWhile
 :: If -v*y1<0 Then ::-v*y1+phi ::Else ::-v*y1 ::EndIf
 :: EndFunc
```

Abb. 10

Die drei Wissenschaftler, nach denen das RSA-Verfahren benannt ist, Ron Rivest, Adi Shamir und Leonard Adleman, wählten in ihrem bahnbrechenden Aufsatz „A Method for Obtaining Digital Signatures and Public-Key Cryptosystems“ (1978) als Klartext das Zitat, das Shakespeare dem Julius Cäsar in den Mund legte:

```
its all greek to me
```

Diesen Text kodierten sie nach der Stellung im Alphabet in eine Klartextzahl, für Zwischenräume setzten Sie jeweils Nullen. In Viererblöcken geschrieben lautet die Nachricht:

0920’1900’0112’1200’0718’0505’1100’2015’0013’0500

Für die RSA-Verschlüsselung wählten Sie die Primzahlen p = 47 und q = 59, als öffentliche Verschlüsselungszahl wählten sie e = 17.

Wie lautet die RSA-Verschlüsselung?

Wir führen die RSA-Verschlüsselung zuerst mit dem Voyage™200 durch. Als Modul erhalten wir n = p·q = 2773. Mit der Funktion `mod( )` kann verschlüsselt werden:

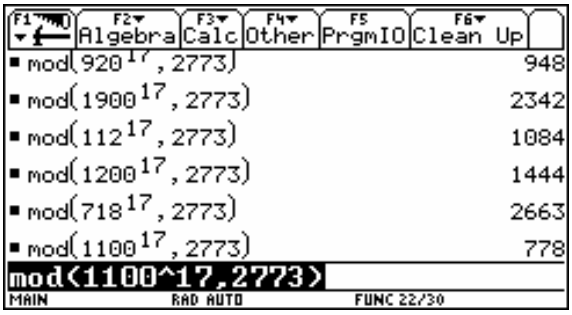

Abb. 11

Nun berechnen wir für phi = (p-1)·(q-1) = 2668 und e = 17 mit `privkey( )` den privaten Schlüssel d = 157 und entschlüsseln die Nachricht:

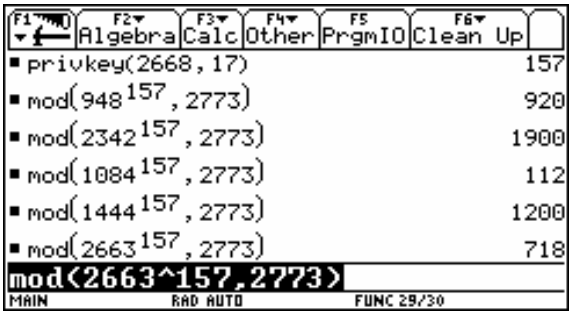

Abb. 12

Beachten Sie: Die Zahl 2663^{157} hat 538 dezimale Stellen (!), eine tolle Leistung, dass der Voyage™200 den Rest dieser Zahl richtig berechnet.

Mit dem TI-83/84 Plus muss jeder Viererblock mit dem Programm `modexp` verschlüsselt werden:

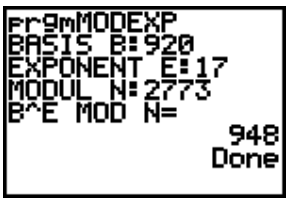

Abb. 13

Mit dem Programm `privkey` wird d berechnet, dann kann wieder mit dem Programm `modexp` entschlüsselt werden:

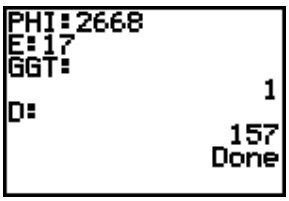

Abb. 14

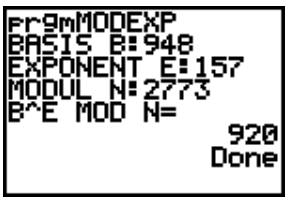

Abb. 15

Ganz herrlich lässt sich mit einem CAS wie TI InterActive!™ verschlüsseln:

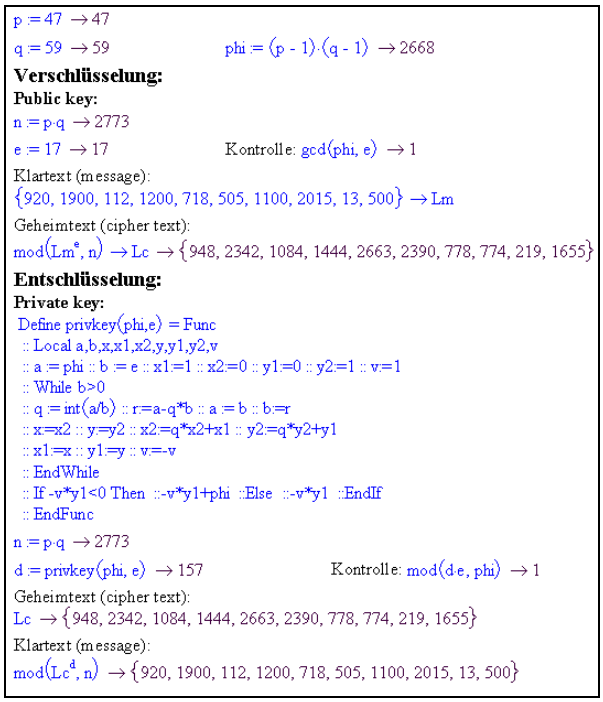

Abb. 16

Übrigens, der derzeit (Juni 2006) kleinste nicht faktorisierte Modul ist RSA-704 mit 212 Dezimalstellen:

74037563479561712828046796097429573142593188889231289084936232638972765034028266276891996419625117843995894330502127585370118968098286733173273108930900552505116877063299072396380786710086096962537934650563796359

RSA security zahlt für die Faktorisierung $30 000. Viel Spaß bei der Faktorisierung!

Wer Genaueres über die zahlentheoretischen Hintergründe wissen will, kann das in ausgezeichneten Büchern nachlesen:

[1] Buchmann, J.: *Einführung in die Kryptographie*; Springer 2004.

[2] Ertl, W.: *Angewandte Kryptographie*; Hanser Verlag 2003.

[3] Schneider et al.: *Mathematik III HAK*; Trauner-Verlag, Linz 2006 -- ein Lehrbuch mit Übungsaufgaben zum RSA-Verfahren

Autor:
Dr. Markus Paul, Innsbruck (A)
markus.paul@utanet.at